



3

Implementation of Industrial Internet of Things and Cyber-Physical Systems in SMEs for Distributed and Service-Oriented Control

Rafael A. Rojas and Manuel A. Ruiz Garcia

3.1 Introduction

The trend toward Industry 4.0 intends to populate traditional shop floors with digitalized systems that are able to share their process parameters, their operative status and express their availability for collaboration with other machines or workers. In other words, this new industrial philosophy foresees each instance of the value chain of a product as a smart one, that is, endowed with decision-making capabilities and the means of communication for valuable information sharing between instances (Kagermann et al. 2013). In this sense, the situational awareness of the manufacturing environment greatly relies on connectivity solutions, like IoT or Internet of Services (IoS) (Gilchrist 2016).

R. A. Rojas (✉) · M. A. Ruiz Garcia
Faculty of Science and Technology, Free University of Bozen-Bolzano,
Bolzano, Italy
e-mail: rafael.rojas@unibz.it

M. A. Ruiz Garcia
e-mail: ManuelAlejandro.RuizGarcia@unibz.it

Although comparable connectivity solutions are not new, thanks to the qualitative change in the processing capacity of modern embedded systems (ES) and the introduction of CPS (Lee 2008) a new kind of networked control system for factory automation is now possible. When integrated in a connected manufacturing environment, CPS replace the traditional automation pyramid and, by combining IoT and automation systems, merge two domains that had been traditionally been separate in industrial systems (Monostori et al. 2016): the IT domain and the operation technology (OT) domain. The former, related to the processing of data to obtain valuable information. The latter, related to the support of physical value creation in manufacturing processes. This integration could be achieved through the digital integration of traditional software systems as enterprise resource systems (ERP) and manufacturing execution systems (MES) with CPS. Following this idea, CPS will become the building blocks of the smart factory, the central CPPS of Industry 4.0.

The major challenges of achieving digital integration are related to the natural software heterogeneity in industrial systems. The effectiveness of modern enterprises depends on hundreds if not thousands of custom-built digital applications that can be acquired from a third party belong to a legacy system or a combination thereof. In fact, programming business applications is a challenging task and creating a single application capable of running a complete business is next to impossible (Hohpe and Woolf 2004). Although ERP systems are the most popular integrations points, they only provide a fraction of the functionalities required in an enterprise. Regardless, heterogeneity of components in modern industrial systems is a necessary or even favorable condition (Lin and Miller 2016). For example, acquiring components produced by different vendors may exploit the benefits of each distributor. Also, the continuous evolution of technology introduces new components that need to be integrated with legacy systems. Moreover, different norms and standards may require specific solutions that are not scalable or convenient for adoption in every application.

This chapter is devoted to the integration between OT and IT, in terms of the commutation network infrastructure necessary to define an IoT industrial solution. In particular, we present the design

tools of a manufacturing service bus (MSB) to overcome many of the aforementioned integration issues, that is, the software infrastructure defining a homogeneous information channel among disparate, possibly event-driven, platforms.

3.2 Fundamentals of Connectivity

3.2.1 The OSI Model

The open systems interconnection reference model (OSI model) defines a seven-layer framework to describe the information flow between digital systems (Zimmermann 1980). It was developed to introduce a mechanisms abstraction to transfer information between a pair of digital systems. This model allows the commutation logic to be decoupled from the actual implementation, which is subject to the particularities of each application.

Figure 3.1 shows how the information flow between a CPS and other system travels through the OSI layers to reach both ends. Each block represents a digital system. At the bottom of each block, it is possible

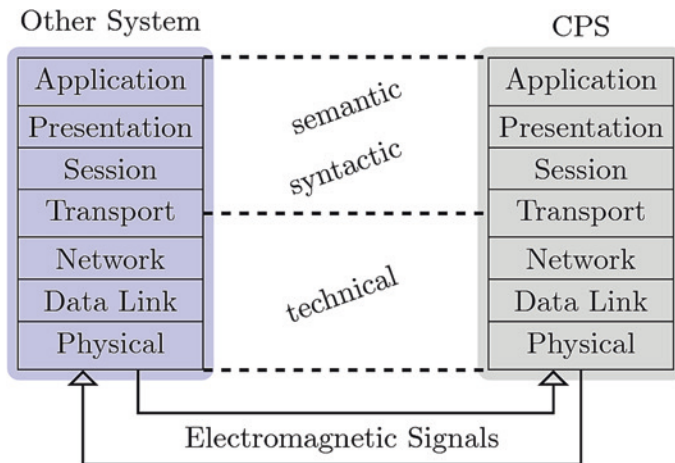


Fig. 3.1 The OSI model

to observe the physical layer, or layer 1, which describes the underlying mechanical, physical, optical, or electrical platform of the communication channel. On top of it, the data link layer or layer 2, provides the means to link two directly connected nodes, to access the physical communication medium and check possible errors induced by it. Next, the network layer, or layer 3, resolves the complex paths that may exist between the origin and destination nodes, thanks to specified target addresses. Layer 4 or the transport layer, sets up an end-to-end connection providing a means of transferring data sequences from a source to a destination host regardless of the routing paths. Layers 5 and 6 are often subject to criticism and are not of interest in this chapter. However, they are often integrated inside the application layer or layer 7, which is of special interest as represents the interface between the software applications running on the digital system and the communication system.

For this reason, the application layer encloses the major challenges of seamless integration between heterogeneous digital systems. On the one hand, this is because implementations of layers 1–4 are rapidly converging to ethernet-based TCP/IP technologies. On the other, it is because each digital system provides its own abstract representations of common sources of information and data therein. For example, consider a hybrid manufacturing station where a collaborative manipulator is integrated with a 3D camera to keep track of the human operator's activities. The manipulator receives data directly from the 3D camera to monitor the pose of the operator, so to collaborate with him in avoiding collisions. Both systems are attached with a direct communication channel based on USB or ethernet. As these digital systems are produced by different manufacturers, they offer different software abstractions of the information of common interest, i.e., the pose of the operator. Such data may be defined, for example, in different units or data structures inside each system. Although some standards may be implemented in the communication channel made available by the manufacturer of the 3D camera, there is a limit in the level of detail that a standard can offer. Moreover, the software inside the robotic application may be designed to represent the position of the operator in a way convenient to the programmer or follow the company's internal standards. Therefore, the data of the

3D camera shared through the communication channel cannot be used without the necessary representation transformation.

3.2.2 CPS Architecture

Heterogeneity is a fundamental characteristic of CPS. In fact, they are composed of three fundamental layers (Sztipanovits et al. 2012) that provide a division of their heterogeneous components. First, the physical layer refers to the material components and their physical interactions. The platform layer refers to the hardware electronics supporting the digital systems comprising the communication infrastructure. Finally, the software layer comprises the operating system and the different digital processes which actually control the CPS and provide means to implement intelligent or complex tasks. The relation between these layers and the OSI model is represented in Fig. 3.2.

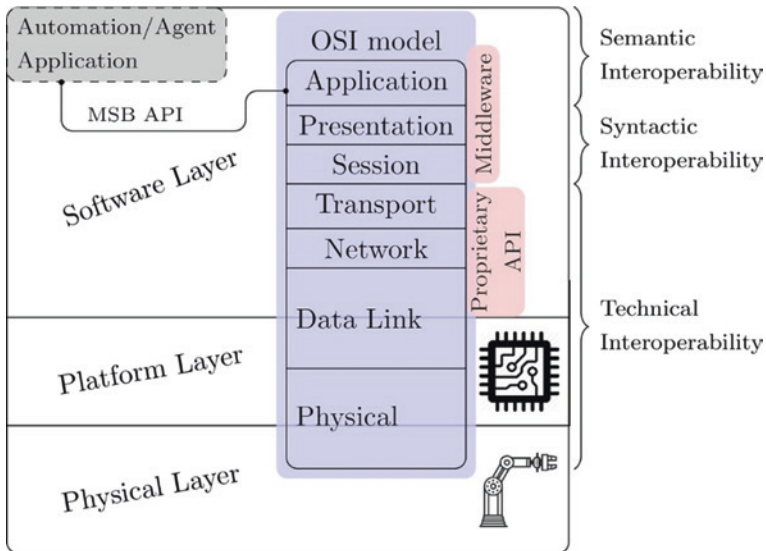


Fig. 3.2 Relation between the layered architecture of CPS and the OSI model

CPS are part of the common trend of pervasive computing, where distributed computing systems represent a dominant paradigm (Wooldridge and Jennings 1995). This concept has been tackled by the idea that intelligent behaviors “emerge” from the interaction of many simple entities. Together, the concept of CPS and “emergent” smart behaviors resonate with the idea of CPPS: a body of autonomous entities that smartly interact to achieve global objectives. Two main paradigms have been proposed to enable smart behaviors of autonomous entities in production systems: multi-agent systems (MAS) and holonic manufacturing systems (HMS). Agents and holons can be defined as self- and ambient-aware entities that can adapt to ambient variations, exhibit goal-oriented behaviors and interact with their peers.

MAS were proposed in the field of artificial intelligence (AI) to characterize such distributed computing systems. An agent may be defined as a system that is situated in some environment, capable of exerting autonomous actions on such an environment to meet its design objectives (Wooldridge and Jennings 1995). For leveraging of such characteristics, authors in Vogel-Heuser et al. (2015), Ji et al. (2016) and Monostori et al. (2016) propose the MAS technology as the main enabler of smart collective behaviors in CPPS. Beyond the fact that agents do not necessarily have a physical part, we desire to underline some important features which CPS share with agents in MAS (Weiss 1999): (i) are self-aware and ambient-aware, (ii) react in a timely way to ambient variations, (iii) exhibit goal-oriented behaviors, and (iv) interact with peers. On the other hand, HMS are constituted by autonomous entities that interact through a variety of hierarchic or egalitarian relations to achieve similar objectives of CPPS. In contrast to MAS, rooted in AI methods to achieve smart emergent behaviors, HMS is a conceptual paradigm motivated by the need to optimize manufacturing systems.

It is worth noticing that both paradigms share the common vision of manufacturing systems defined in terms of autonomous and cooperative units. As a consequence, the integration of a heterogeneous digital system is, in turn, an enabler of smart behaviors in CPPS. Therefore, as depicted in Fig. 3.2, the software layer integration represents a key

milestone to enable the “agent behavior” of the CPS regardless of the underlying data heterogeneity contained inside the CPPS.

3.2.3 The Concept of Interoperability

In our context, integration implies a set of steps allowing a body of disparate systems to be treated as a whole (Bellman and Landauer 2000). As such, this conglomerate unicum of entities can be understood, monitored, reasoned about, configured or controlled without requiring explicit knowledge of its enclosed systems. As remarked upon in Gössler and Sifakis (2005), achieving integration requires that the system meets the following conditions: (i) compositionality, that the behavior of the system is predictable from the behavior of its components, (ii) composability, that the attributes of each component do not depend on other components nor on their interactions. Among all concerns that must be addressed to integrate heterogeneous systems, in this chapter, we are only interested in those related to the information sharing between software layers of different CPS. Under this delimitation, by integration of CPPS, we refer to the necessary steps to create a coherent and seamless information exchange between CPS software layers (Vernadat 2007). In other words, we limit our analysis to the interoperability aspects of the integration.

Interoperable systems provide understood interfaces for message exchange and functionality sharing. Therefore, three levels of interoperability can be identified: technical, syntactic, and semantic. Technical interoperability represents the capacity to exchange a raw sequence of bites. Syntactic interoperability is associated with data formats and structures, i.e., the symbols represented by such sequences of bits. Finally, semantic interoperability is the capacity to exchange meaning between systems. As a consequence, semantic interoperability necessarily depends on syntactic interoperability, which in turn, depends on technical interoperability. Unlike the wide concept of integration, interoperability is related to the coherence and uniformization of data and is a prerequisite to integrability itself. In fact, integration must assure composability, compositionality, and flexibility (Lin and Miller 2016).

3.2.4 Loosely Coupled Systems and SOA

Achieving loose coupling is to reduce the assumptions two software applications make about each other when they exchange information. The more assumptions they make about each other, the less tolerant is the connectivity solution to changes in the system. Common assumptions that lead to a tightly coupled system are (i) about the platform technology (internal bit representations), (ii) location (hard-coded addresses), (iii) time (availability), and (iv) data formats.

Integration of CPPS becomes familiar with a long history of effort to integrate disparate digital platforms that begins when computers and software applications become pervasive in office and business (Chappell 2004). In these early days, it was noted that a large number of small distributed software procedures allow for flexibility and reuse. Those approaches achieved simple remote communications by packaging a remote data exchange into the same semantics as a local method call (a traditional function of programming languages). This strategy resulted in the notion of a remote procedure call (RPC). Common implementations of RPC are CORBA, Microsoft DCOM, .NET Remoting, Java RMI, XML-RPC and SOAP. This marked the evolution from point-to-point integration solutions to the so-called service-oriented architectures (SOA) (Chen et al. 2008). Service is a common name for a functionality that is executed in distributed systems and SOA is an approach to encapsulate functional components in services. However, one of the main challenges to implementing a SOA is that remote communication invalidates many of the assumptions that a local method call is based on. To achieve a well-designed SOA requires management, a centralized service directory and effective documentation.

In parallel, communication solutions in industrial automation systems also began by using point-to-point solutions that were expensive and bulky (Felser 2002). In the two “worlds,” business/office and industrial automation, the following metaphor appears as the ideal solution to achieve seamless connectivity: a single cable or bus of communication where a message can transit and every communication transaction may appear as a point-to-point exchange of data.

In the business/office realm, the term ESB was introduced to describe the software infrastructure capable of emulating such a bus and, in the field of industrial automation, the term “fieldbus” was coined. Unlike ESB, fieldbuses were originally developed for ES with reduced resources designed to exchange small amounts of data with stringent timing requirements. With the introduction of CPS factory shop floor, the quality of the exchanged information becomes more similar to office systems. Instead of exchange positions of servos, a CPS may receive the complete model of a product to be assembled.

3.2.5 The Publish and Subscribe Pattern

“Publish” and “Subscribe” is a kind of messaging pattern designed to achieve a network of loosely coupled nodes. To reduce the number of assumptions made about peers, messages are sent through specialized channels often called topics. The middleware infrastructure hides the details of complex message routing between nodes and permits access to a channel using a semantic identifier instead of complex addresses such as IP number and port. To describe the type of messages that a channel conveys and which channels are available in the systems, it is possible to implement centralized systems to retrieve the desired information. Thanks to this middleware, it is possible to send messages only assuming that the receiver is listening at a specific channel. Nodes which send messages through a channel are called publishers of that channel. On the other hand, nodes which listen to a channel are called subscribers of that channel. Generally, channels have only one direction, but several implementations are possible. Publishers do not program messages to be sent directly to specific subscribers, but instead characterize published messages into classes using the available channels without making any assumptions about the subscribers’ routing requirements. Similarly, subscribers express interest in one or more classes or channels and only receive messages that are of interest, without knowledge of which publishers, if any, there are.

Often publish-subscribe systems are constructed around a central entity called a broker that is responsive to managing channels

and its publishers and subscriber. Brokers contain the list of the available channels and the IP addresses of every node in the publish-subscribe network. Each node performs client-server style operations with the broker to subscribe to channels or create new channels to publish data. Publish-subscribe is a sibling of the message queue paradigm, and is typically one part of a larger message-oriented middleware system. As publisher can “publish” information without regard for subscribers and a subscriber can “subscribe” to information without regard for publishers; the result is a loosely coupled network where each node can be replaced independently of one another. This kind of messaging patterns is well suited to exchanging data updates between components and allows the communication paths to be optimized based on their requirements. It provides scalability for an evolving number of data sources and contributes to IIoT reliability, maintenance, and resilience by the decoupling of publishing and subscribing components in both location (location transparency) and time (asynchronous delivery). This decreases the likelihood of fault-propagation and simplifies incremental updating and evolution. The concept of channels allows flexibility in the interaction between nodes allowing periodic (time-driven) or responsive (event-driven) behaviors depending on the needs of the user. Asynchronous data transfers can be implemented in different publish-subscribe solutions making the IIoT more robust to component failures and unexpected delays.

3.2.6 Service Discovery, Zero Configuration, and Plug-and-Play/Work Networks

IIoT requires a flexible method for service composition, such that different functionalities can be dynamically integrated at run-time. Therefore, allowing dynamic networks to address services, without affecting the end users, represents a desirable quality. In this regard, the numerical-based addressing mechanisms used by IIoT platforms have two principal drawbacks. First, they are intrinsically not human-friendly. Second, IIoT networks require dynamic address assignment on hardware variations or software migrations. To overcome such limitations, a stable name can be

associated with each service through a uniform resource identifier (URI), a string of characters with a predefined set of syntax rules that unambiguously identifies a particular resource. A URI does not refer to a particular piece of hardware or software, but a logical service with which any end user can communicate using a specified protocol. In the past, human-meaningful names (URI) were bound to computer-meaningful network addresses by manual configuration of the network. Today, there are several technologies to automate this procedure allowing each network entity to be interrogated about its services and protocols. As a result, any device or application can share its virtual representation or manifest (Monostori et al. 2016). In the context of desktop computers, the ability to automatically add devices to a network without having to manually register its services and configuration has been called zero configuration networking (Steinberg and Cheshire 2005). Zero configuration networking is based on the following three elements:

- Automatic IP address selection for networked devices (without DHCP server).
- Translation between names and IP addresses without a DNS server (Multicast DNS).
- Automatic location of network services through DNS service discovery (which eliminates the need for a directory server).

On the other hand, in the context of manufacturing, the term *plug-and-work* appears with the capability of a production system to automatically identify a new or modified component and to correctly integrate it into the running production process without manual efforts and changes within the design or implementation of the remaining production system (Schleipen et al. 2015). Schleipen et al. (2015) identify the following requirements for the application of the plug-and-work technology:

- *Component description*: the ability to get a complete description of an entity in the system.
- *Component selection*: the ability to compare all entities capabilities and choose the one which is able to perform some task.

- *Component access*: the ability to communicate with the entity.
- *Component control*: the ability to provide a control structure to the entity.

Under the perspective of IIoT, we recognize both concepts of zero configuration networks and plug-and-work as different names for a method of achieving dynamic CPPS composition. This composition is characterized by a loose coupling between the SOA and messaging model (OSI layers 4–7) and low-level network implementation (OSI layers 1–3). In general, the zero configuration infrastructure should be agnostic to application protocol design and advertise any kind of application protocol. At the lowest level, zero configuration networks or plug-and-work IIoT may be achieved through the implementation of the following technologies:

- *mDNS*: Multicasting DNS protocol resolves host names to addresses within small networks that do not include a local name server. It uses IP multicast user datagram protocol (UDP) packets.
- *UPnP*: Universal plug-and-play is a set of networking protocols on the top the internet protocol (IP), leveraging on HTTP to provide device/service description, actions, data transfer, and eventing. Device search requests and advertisements are supported by running HTTP on top of UDP (port 1900) using multicast (known as HTTPMU).

However, it is worth noticing that application-layer protocols are subject to constant evolution and changes.

3.2.7 Ethernet-Based Connectivity Technologies for SME

IT system integration costs may be a factor able to outweigh all other technical considerations (Wagner et al. 2015). Such costs include materials, software licenses, hardware installation, and other technical labor. Although there are some efforts of providers to adapt their offerings to the needs of small- and medium-sized enterprises (SMEs),

usually their primary customers are large organizations able to afford such costs (Cruz-Cunha 2009). However, in the case of SMEs, the deployment of IIoT requires considering several trade-offs. Among others, we highlight the trade-offs between outsourcing and in-house software development and between open-source software (OSS) and proprietary software tools. OSS is a viable alternative in terms of costs and independence to vendor-locked applications (Olson et al. 2018). In fact, OSS does not require vendor-specialized consultants and tools, and allows the use of in-house available hardware and software rather than proprietary ones, subject to maintenance contracts. According to Weber (2004), OSS enhance software reliability and quality through independent peer review and rapid evolution of source code. Following these ideas, together with Lin and Miller (2016) and Gilchrist (2016), we suggest that OSS and open standards are also key enablers of IIoT in SME. To support this statement, we observe that SMEs have a relatively small amount of digital systems and processes to be integrated, thus they may be able to afford to develop in-house software applications. Moreover, thanks to the source code availability, OSS gives SMEs the possibility of developing highly customized and lean solutions based on their know-how. As a last remark, we emphasize that a growing amount of companies are providing support to open source IT solutions (Olson et al. 2018).

The following publish-subscribe and SOA messaging protocols, implemented on top of the TCP/IP protocol, are useful OSS for IIoT applications:

- Java message service (*JMS*), part of the Java EE, defines a generic and standard application programming interface (API) for the implementation of message-oriented middlewares. It does not provide any concrete implementation of a messaging engine.
- The advanced message queuing protocol (*AMQP*) is an open standard application-layer protocol for messaging and publish-subscribe messaging. *AMQP* is often used with *RabbitMQ*, a free and complete *AMQP* broker implementation and API for Java, C# and Erlang.
- *RabbitMQ* is written in Erlang and it is available in Ubuntu through the *rabbitmq-server* package. It offers a flexible broker implementation based on open standards.

- Message queue telemetry transport (*MQTT*) is an ISO standard (ISO/IEC PRF 20922). It is broker-based and designed for networks with limited resources.
- Extensible messaging and presence protocol (*XMPP*) is a communication protocol for message-oriented middleware based on XML (extensible markup language).

3.3 The Integration Drivers

The challenges of CPPS integration span far across operational and technical issues. In fact, beyond classical interoperability between digital systems, IIoT also implies the integration between organizational units and IT systems. Such an integration may require a significant shift in corporate politics and defining clear separations between inherent modules is not an easy task. In Lin et al. (2015) a set of different viewpoints are given to achieve such a modularity. Following this idea, we distinguish between organizational and technical drivers for the IIoT integration effort. On the one hand, organization drivers refer to those required for OT, i.e., to create physical value for a global market. On the other, technical drivers are those localized at the digital platforms, specifically related to the available or required digital systems.

3.3.1 Organizational Drivers

To define the necessary information flows inside a CPPS, it is also necessary to map business/organization processes into data requirements (see Fig. 3.3). The business vision, values, practices, key objectives, and capabilities are fundamental inputs to clearly define such system requirements. As a consequence, it is of fundamental regard to identify the OT activities, their inherent tasks and the roles of each comprising party on the accomplishment of such activities. Either tasks dependencies, constraints, and workflows need to be mapped into data requirements and data flows.

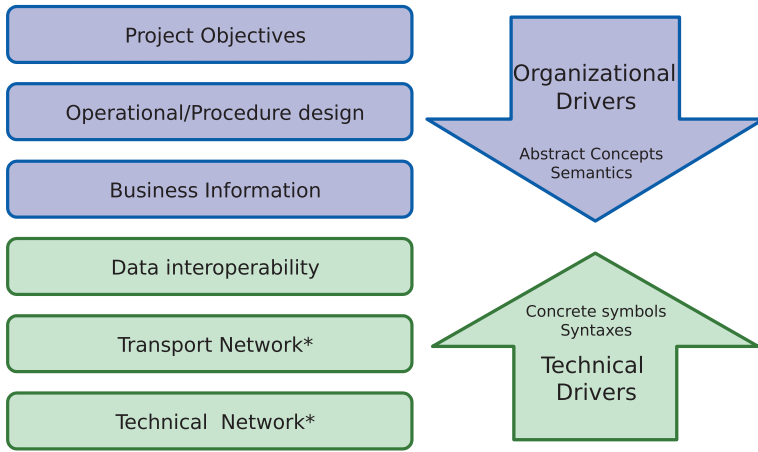


Fig. 3.3 The drivers of the integration

In Lin et al. (2015), the concept of domains is introduced, to simplify the functional separation between building blocks of business and organization processes. Each functional domain lies on a specific granularity and time scale, and can be hierarchically organized from high-level intelligence, to low-level control. The control domain directly deals with low-level OT. The operations domain contains the functional elements enabling operability of the hardware on the control domain (monitoring, register, track, deploy, and retrieve assets). The information domain provides the collection of data from all domains to high-level analysis. The application domain applies and defines coarse-grained logic, rules, and models for workflows. In other words, it provides a high-level abstraction of functionalities that could lie in a specific domain or be distributed among many of them.

3.3.2 Technical Drivers

The implementation viewpoint of Lin et al. (2015) is concerned with the technical representation of the IIoT. This representation takes into account the technologies and system components required to

implement all activities and functions prescribed by the usage and functional viewpoints. Such viewpoints address all technical issues associated with the future use of the system, how to enable those usages through specific functions and how these functions interact. It might also be observed that the development of integration solutions is constrained to the limited or almost nonexistent level of customization that participating applications may have. In most cases, applications belong to external providers or legacy systems and cannot be changed or upgraded. This often leaves the integration developers in a situation where they need to make up for deficiencies or idiosyncrasies inside the participating applications. Often, it would be easier to implement part of the solution inside the application endpoints, but for political or technical reasons that option may not be available.

3.4 Connectivity Architecture

3.4.1 Ethernet-Based Automation System

In the last decade, the ethernet has become increasingly popular and pervasive inside the mid-level of the automation pyramid (Sauter 2014). Ethernet implements the OSI layers 1 and 2 and it is generally deployed with the complete IP protocol stack (OSI layers 3 and 4). Such a combination of ethernet and IP provides important building blocks to achieve technical interoperability between disparate digital systems. In fact, the ethernet is becoming the de facto standard in office, enterprise or business systems and modern CPS. However, the ethernet—as it is known in office or business environments—could not meet the requirements of industrial automation, since it lacks determinism and it has not been designed for real-time applications. Although the introduction of switched ethernet, megabit and gigabit ethernet has notably mitigated such problems, some practical realization of ethernet-based networks in industrial environments still needs special care. The IEC 61784-2 standard introduces terms like “industrial ethernet” or “real-time ethernet” to define a set of ethernet-based technologies able to meet strict industrial and technological requirements.

To address the requirements of industrial level ethernet systems, we follow Kim et al. (2014) by differentiating the types of data transmitted in an IIoT system into *configuration data* and *process data*. Configuration data defines the set of parameters required to configure the system, including remote management and operations work-flow. Process data identifies the process states, thus, depending on the process nature, this type of data may be periodic (when it is constantly generated) or aperiodic (when it is generated sporadically). To illustrate those types of data we can consider a classical robotic pick and place operation, where objects are moved from one place to another and sensors detect the availability of those objects to be placed. Both the measured positions of the joints and the commands to move them are periodic process data. They are generated constantly and describe part of the current state of the process. Also, the information generated by sensors falls into the category of process data. On the other hand, the program controlling the robot during the task execution belongs to a set of configuration data.

Each type of data flow has its own qualitative requirements, update rates, and tolerated latency times. In particular, we can distinguish between the following latency categories:

1. Human-control systems, where humans are involved in the system observation and the characteristic times are of the order of 100 ms.
2. Process control systems, which relate to computer numerical control (CNC) and programmable logic controller (PLC) systems and the characteristic times are of the order of 10 ms.
3. Motion control system, where the timing requirements are less than 1 ms. Motion control latency requirements are also called real-time.

Human and process control systems may be designed using slightly customized ethernet-based technologies. Among others, technologies like Profinet, TCnet, and Powerlink are able to meet process control requirements. Real-time networking solutions (of the order of 1 ms) are too restrictive to provide the level of flexibility that IIoT requires. In fact, CPS internal control mechanism needs to be endowed with enough levels of autonomy to avoid streaming control data flows through the network (goal-oriented control).

3.4.2 A Layered Design for Manufacturing Service Bus

Although it has pertinence in describing a communication system, the OSI Model is inadequate for representing the features of an IIoT system in a convenient way. As we have noticed before, the OSI model consists of many layers that, in general, are defined by dominant communication technologies, as ethernet and derivatives. Following Lin et al. (2015), we aggregate the functionalities of the OSI layers into two separate layers. The OSI layers 1 (physical) through 4 (transport) are collected in the *communication transport layer*, to allow the basis for technical interoperability to be addressed along with the ability to reach endpoints along structured networks. On top of the communication transport layer, we define the *connectivity framework layer*. Such a layer spans the functionalities of the OSI layers 5 (session) through 7 (application) and provides the software infrastructure where middlewares are placed. The syntactic interoperability channel (with common and unambiguous data formats) and the ability to create messages without considering the particular endpoint implementations are placed inside this layer. The connectivity framework layer also provides the service discovery functionalities, the resources access handling, the high-level data exchange patterns (peer-to-peer, client-server, publish-subscribe, etc.), the means of security and the interoperability model in different programming languages.

Such subdivision of the connectivity functionalities is of particular benefit for ethernet-based IIoT systems. This comes from the fact that such a subdivision reduces the degrees of freedom of the OSI model and decouples two realms addressing different functional duties inside any ethernet-based systems. We call the collection of the functional components, given by the combination of these two layers, the MSB. This term was coined as an equivalent to enterprise service bus (ESB) (Monostori et al. 2016) in manufacturing systems. However, an MSB is differentiated from an ESB in terms of the quality and complexity of the operation that it can perform over the transit of data it handles. The MSB provides an interface between the low-level fieldbus devices, the CPS, and the business systems, generally based on an ESB and relying on the ISA-95 standard.

This solution allows the IT landscape of the factory floor to be integrated with enterprise-level IT systems. Similar solutions have been addressed by several authors. In MESA (2008), the concepts of SOA are widely explained from the perspective of manufacturing systems and an architecture for MSB is presented. The European Innovation Project SOCRADES (De Souza et al. 2008) presents an MSB system based on web services. The European Innovation project IMC-AESOP (Colombo et al. 2014) proposes an integration approach for CPPS based on cloud technologies. Minguez et al. (2010) explored the concept of MSB from a general perspective. Other proposals may be found in Zhang et al. (2018).

3.4.3 Physical and Logical Network Topologies of the MSB

To achieve coordination and orchestration of a distributed body of CPS, it is mandatory to have a correct structure and organization of the communication functions. For this reason, we address the physical topology issue of the communication hardware through the introduction of a three-tier architecture, sketched in Fig. 3.4.

This architecture collects all nodes of the CPPS's network into three tiers in accordance with its functionalities and requirements. Such tiers are hierarchically structured and each one contains qualitatively different holons or agents. At the lowest tier, almost every agent has a physical part, i.e., a CPS that enables the creation of physical value in implementing OT. At the upper tiers, most of the agent has only a digital part implementing IT for data processing, and retrieving valuable information for high-level decision-making. At the lowest level, we have the edge tier, which implements most of the control-related systems necessary to implement the OT directly in the physical world. In this tier, a proximity network connects sensors, actuators, CPS, and other elements in a bunch of heterogeneous communication technologies which are connected to the same baseline. Such a baseline, also called a *backbone network*, should be based on ethernet and represents the gateway between the shop floor and the office/enterprise networks.

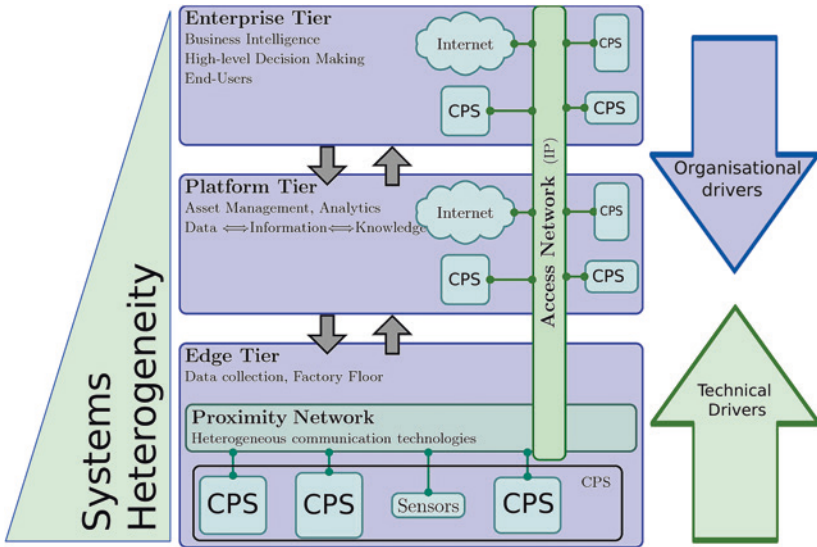


Fig. 3.4 The three-tier architecture

The platform tier implements functional capabilities to enable operability of the hardware and collects information from all instances of the CPPS, providing high-level analysis and intelligence about the overall system. It implements most of the information and operational domains and is composed by digital systems which lie somewhere between desktop computers and customized industrial computers.

The enterprise tier implements most of the business domain functionalities, data analytics and high-level decision-making and is completely composed by desktop-type computers. In our model, the last two tiers are connected to the edge tier through the so-called access network, which, contrary to the proximity network, does not have the same constraints that can be found on the factory floor and connects qualitatively different systems.

It is worth noticing that this architecture not only splits functional components in accordance with the necessary technologies for communication. Indeed, it also splits the decisional task into different levels. At the enterprise tier, strategic and high-level decisions, which are

characteristic of the business functional domain, are taken. At the platform tier, the granularity becomes thinner, and more technical aspects are considered to introduce commands into the system. The three-tier architecture proposes a specific kind of network for each tier suitable for the type of digital system lying on the tier. Every actor in a CPPS is represented by a node in some tier and, at the same time, a holon with a digital part associated with such a node.

3.5 Case Study

3.5.1 The Smart Mini Factory

The Smart Mini Factory laboratory of the Free University of Bozen-Bolzano, founded in 2012 and focused on the research area Industrial Engineering and Automation (IEA), aims to reflect the principles of lean and agile production on a small and realistic scale. Inside the Laboratory, three main activities take place: applied research, focused on industry-driven use cases; teaching activities, as part of the bachelor's and master's program for industrial and mechanical engineering; seminars for industry on all aspects of Industry 4.0, to facilitate the transfer of knowledge from research to industry. To support and boost these activities, the Laboratory is equipped with different robotic platforms and devices:

- Adept Cobra i600: four axis manipulator with a SCARA base and one additional wrist joint.
- Adept Quattro s660H: four arm delta robot designed for industrial high-speed applications like packaging.
- Adept FlexBowl: rotary feeder for a wide range of loose parts.
- Two Basler scout giga-ethernet camera.
- Robotiq 85 and 140 adaptive grippers controlled via modbus RTU using RS-485.
- ABB IRB 120: compact anthropomorphic robot able to handle payload of up to 3 kg with a reach of 580 mm.

- KUKA KMR iiwa: combines the strengths of the sensitive LBR iiwa lightweight robot with those of the KMR mobile and autonomous platform.
- Universal Robot UR3: 6-rotating-joint anthropomorphic manipulator suitable for light assembly and high precision tasks.
- Universal Robot UR10: 6-rotating-joint anthropomorphic manipulator suitable for heavier-weight process tasks.
- Ulixes Der Assistent A600: manufacturing assistant system for manual assembly station based on a projector and a visual tracking system.

Since all robots and devices are equipped with an ethernet interface, a proximity network connecting them through a single ethernet baseline has been defined. This network defines the support of the communication transport layer of our MSB (see also Fig. 3.5).

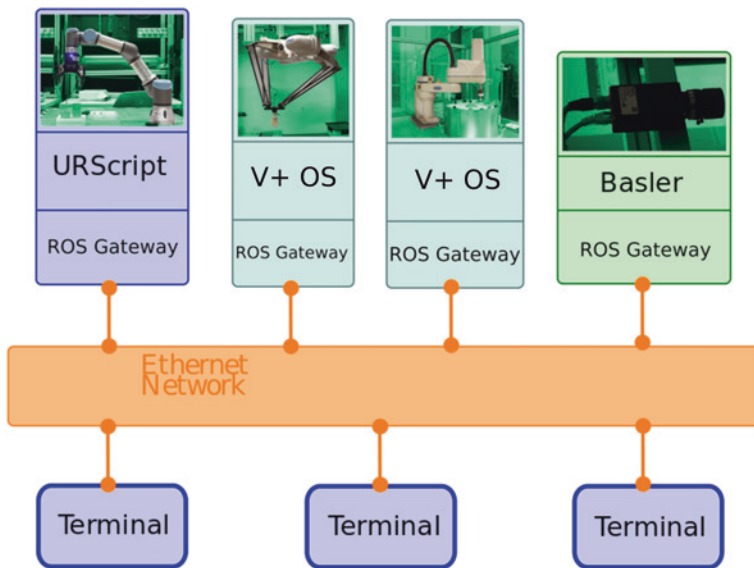


Fig. 3.5 Sketch of the mini-factory network (Reproduced with permission from Smart Mini Factory Lab, unibz)

3.5.2 Design of the Manufacturing Service Bus

Following the layered design guidelines defined in Sect. 3.4.2, we decouple this task in the design of two separate layers: the connectivity framework and transport framework. For the transport network, we choose switched ethernet IP with TCP or UDP protocols at the OSI layer 4. On the other hand, for the connectivity framework layer, we implement the robot operating system (ROS) communication systems. Our MSB should provide the following features common for ESB (Menge 2007):

1. *Invocation*: each software layer will be able to call on the services available in other platforms.
2. *Mediation*: capacity to translate between different data formats (syn-tactic interoperability).
3. *Adapters*: API wrappers to make each system resource available to the network.
4. *Management*: a logging system for control of process, auditing, etc.
5. *Asynchronous Messaging*: ability to send and receive messages without the need for explicit coordination with the peer.

The invocation and asynchronous messaging features represent the interface between the MSB and the agent application of the CPS. Such an implementation on the native programming language of the CPS is called the MSB API, and should also be available to other end users of the CPS's software layer. To understand the extent to which the MSB may be implemented on the CPS's software layer, it is necessary to introduce the concepts of protocol data unit (PDU) and service access point (SAP) defined in the OSI model. A PDU is a sequence of bits where information is represented using a set of rules to be mutually intelligible by two peer layers. The interface that a lower layer provides to its upper layer for data encapsulation/decapsulation is called SAP.

Each CPS has a vendor-specific software layer that should provide a particular SAP to the programmer. Such an SAP determines the PDU with maximum flexibility available in a CPS. Such PDU should grab

all the requirements for the abstraction of the CPS as a virtual identity that can be addressed from the connectivity framework. To provide a connectivity interface, the CPS manufacturer has to implement an SAP on the top of a standard OSI layer (as TCP/IP connectivity) or on the top of a custom protocol. If the PDU is sufficiently flexible, it would be possible to implement the MSB at the CPS as it may be implemented on a desktop computer. If not, it will be necessary to implement and interface between the CPS and the network.

3.5.3 Connectivity Framework Gateways

Normally, ES have a limited software layer for providing a communication interface directly implemented on the OSI layer 7. That means that commands and configuration functions are directly sent through the communication interface. In such cases, it is not possible to implement an MSB API. On the other hand, several CPS do not provide an SAP with such flexibility to allow an MSB API or it is not convenient. To solve this issue, it is necessary to add the required computational features using the concepts of a connectivity framework gateway and administration shield (Adolphs et al. 2015; Ye and Hong 2019).

Unlike a communication transport gateway, a connectivity framework gateway does not convey PDU into wrappers built into other layers as MODBUS over TCP do. Such gateways provide the architectural construct to incorporate connectivity technologies into a device by conveying the semantic meaning of data through different representations. On the other hand, an administration shield gives an object the means to be considered a CPS. Such an administration shield may be implemented using different single board computers available on the market.

3.5.4 The ROS Protocol

The robot operating system is the result of a communized effort to cope with the challenges and characteristics of developing distributed robotic platforms. It is open-source software (OSS) with a strong community of developers and widely available documentation. In spite of this, its

name, ROS, is not an operating system but a meta-operative system (O’Kane 2014), in the sense that it is a software platform providing an API and a series of tools for developing and handling a distributed peer-to-peer network of processes called nodes. Nodes are processes at the software layer of a CPS as the agent application. ROS provides two elements of interest for this work: a multi-platform API for developing networking applications and a semantic interoperability protocol. The ROS communication system constitutes a connectivity framework layer to the top of the ethernet TCP/IP protocol which provides services, a publish and subscribe protocol and actions. In this chapter, we will only consider publish-subscribe messaging systems and services. The channels of the ROS publish-subscribe systems are called topics and each topic defines the specific data type that it conveys.

Our MSB is a middleware for IIoT where process data is conveyed using messages in a publish-subscribe pattern and configuration data is conveyed using services. The specific data representations are taken from the ROS specification. The ROS network relies on a *master* which acts as the broker in publish-subscribe systems. The master is also a node that provides node registration and lookup for services and topics to the other nodes. The ROS master is basically an XML-RPC server maintaining a list of topics and services with their corresponding linked nodes: publishers and subscribers in the case of topics; providers and clients in the case of services. As a rule, ROS only allows one node to provide one single service. The master node is also responsible for providing a centralized data storage mechanism available to all other nodes of the network. Peer-to-peer data connections between nodes are also negotiated through XML-RPC. Such data connections can be established through TCP/IP or UDP, depending on the particular applicative context. Inside the ROS implementation, publishers act as servers and subscribers as clients. When a node registers a publisher to a desired topic, it sends the XML-RPC call with the URL of its own XML-RPC server together with the target topic name. After the call is received, the ROS master pushes the node’s URL inside the publisher’s list for the given topic and returns a binary status code (failure or success). In the case of success, the publisher node automatically allocates a port to the top of the OSI layer 4 (UDP or TCP) and waits, listening

for incoming connections. When a node registers a topic subscriber, it sends the XML-RPC call with the name of the topic and the underlying message type. In response to this call, the ROS master returns a list of all XML-RPC servers already registered as publishers to the given topic. To connect to a specific publisher, the subscriber must interrogate the publisher node through its XML-RPC interface. The publisher replies to this call with the URL of the allocated port and the communication protocols (UDP or TCP). At this point, the subscriber allocates a socket on top of the OSI layer 4 and, as a client, initiates the connection with the publisher. Services are implemented in ROS in a simple client-server architecture. Service providers are registered through the ROS master's XML-RPC call specifying its own URI. To access a service, a node interrogates the ROS master using an XML-RPC call where the name of the service is specified. As an answer, the master communicates to the service requester the URI of the service provider. At this point, the client initiates a TCP/IP based connection to retrieve the service.

3.6 Conclusions

We presented a framework for the implementation of an ethernet-based IIoT for CPPS in SME. The framework is rooted in the concepts of CPS connectivity and actor aggregation inside a common MSB as a layer for syntactic interoperability of the production system. The main concern was how to enable interoperability between software layers of different CPS. Thanks to the decomposition of the integration effort into drivers and layers, it was possible to restrain the MSB implementation inside the software layer of the CPS. In the particular case of the Smart Mini Factory laboratory, these features were implemented through a wired ethernet network with a broker-based publish-subscribe system of services.

In spite of its effectiveness, this model has some limitations with respect to industrial-scale IIoT systems. For example, the limited number of devices used in this study does not reflect the characteristic of an IIoT system which has thousands of devices. Such a characteristic requires a more detailed analysis of the transport framework and a

larger effort in system integration. On the other hand, thanks to our mild time-delay constraints we could avoid the design of a near real-time channel for data streaming. However, as we are focused on SMEs such a small number of CPS may represent an accurate model. One fundamental limitation of designing an MSB on top of the ROS protocol is given by the need of a master node to route services discovery and resource sharing between nodes. In terms of reliability and resilience, a failure comprising the master node (either in terms of hardware or software) could imply the entire network collapses. Therefore, in industrial environments, a distributed and redundant approach to network resources management and sharing should be preferred. Also, in contrast to broadcast approaches, multicast-based service discovery allows one single data packet to be delivered simultaneously to a group of nodes. Moreover, when wireless networking comes into play, specialized IP mobility solutions are required to handle changes to clients and possibly host locations. Finally, we underline that we did not address our IIoT system from a security perspective (Rehman and Gruhn 2018). Security is a transversal issue in modern IT systems. In regard to CPS, IT-based aggressions may also become physical, implying a coupling between safety and security. Such an important concern requires a dedicated analysis, which is out of the scope of this chapter.

This work presents the first steps for building an IIoT system at the Smart Mini Factory Laboratory. The findings of our research should not only serve as a basis for a further scientific development of CPPS, but also give practitioners an overview of which enabler should be considered in the implementation of Industry 4.0 and especially CPPS in the smart factory of the future.

References

- Adolphs, P., H. Bedenbender, M. Ehlich, and U. Epple. 2015. Reference Architecture Model Industrie 4.0 (rami4.0) (Tech. Rep.). VDI/VDE, ZVEI.
- Bellman, K.L., and C. Landauer. 2000. Towards an Integration Science. *Journal of Mathematical Analysis and Applications* 249 (1): 3–31. <https://doi.org/10.1006/jmaa.2000.6949>.

- Chappell, D. 2004. *Enterprise Service Bus*. Sebastopol: O'Reilly.
- Chen, D., G. Doumeingts, and F. Vernadat. 2008. Architectures for Enterprise Integration and Interoperability: Past, Present and Future. *Computers in Industry* 59 (7): 647–659. <https://doi.org/10.1016/j.compind.2007.12.016>.
- Colombo, A., T. Bangemann, S. Karnouskos, J. Delsing, P. Stluka, R. Harrison, F. Jammes, and J.L. Lastra. 2014. Industrial Cloud-Based Cyber-Physical Systems. The IMC-AESOP Approach.
- Cruz-Cunha, M.M. 2009. *Enterprise Information Systems for Business Integration in SMEs: Technological, Organizational, and Social Dimensions: Technological, Organizational, and Social Dimensions*. Hershey, PA: IGI Global. <https://doi.org/10.4018/978-1-60566-892-5>.
- De Souza, L.M.S., P. Spiess, D. Guinard, M. Köhler, S. Karnouskos, and D. Savio. 2008. SOCRADES: A Web Service Based Shop Floor Integration Infrastructure. In *The Internet of Things*, ed. C. Floerkemeier, M. Langheinrich, E. Fleisch, F. Mattern, and S.E. Sarma, 50–67. Berlin and Heidelberg: Springer. https://doi.org/10.1007/978-3-540-78731-0_4.
- Felser, M. 2002. The Fieldbus Standards: History and Structures. Technology Leadership Day 2002. MICROSWISS Network.
- Gilchrist, A. 2016. *Industry 4.0: The Industrial Internet of Things*. Berkeley, CA: Apress. <https://doi.org/10.1007/978-1-4842-2047-4>.
- Gössler, G., and J. Sifakis. 2005. Composition for Component-Based Modeling. *Science of Computer Programming* 55 (1): 161–183. <https://doi.org/10.1016/j.scico.2004.05.014>.
- Hohpe, G., and B. Woolf. 2004. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Boston: Addison-Wesley Professional.
- Ji, X., G. He, J. Xu, and Y. Guo. 2016. Study on the Mode of Intelligent Chemical Industry Based on Cyber-Physical System and Its Implementation. *Advances in Engineering Software* 99: 18–26. <https://doi.org/10.1016/j.advengsoft.2016.04.010>.
- Kagermann, H., W. Wahlster, and J. Helbig. 2013. Securing the Future of German Manufacturing Industry: Recommendations for Implementing the Strategic Initiative Industrie 4.0. acatech, Final Report of the Industrie 4.0 Working Group (Tech. Rep.). Acatech.
- Kim, J., J. Lee, J. Kim, and J. Yun. 2014. M2M Service Platforms: Survey, Issues, and Enabling Technologies. *IEEE Communications Surveys and Tutorials* 16 (1): 61–76. <https://doi.org/10.1109/surv.2013.100713.00203>.
- Lee, E.A. 2008. Cyber Physical Systems: Design Challenges. In *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time*

- Distributed Computing (ISORC)*, 363–369. <https://doi.org/10.1109/isorc.2008.25>.
- Lin, S.-W., and B. Miller. 2016. Industrial Internet: Towards Interoperability and Composability (Tech. Rep.). Industrial Internet Consortium.
- Lin, S.-W., B. Miller, J. Durand, R. Joshi, and P. Didier. 2015. Industrial Internet Reference Architecture (Tech. Rep.). Industrial Internet Consortium.
- Menge, F. 2007. Enterprise Service Bus. In *Free and Open Source Software Conference*, 2, 1–6.
- MESA. 2008. Soa in Manufacturing Guidebook (Tech. Rep.). MESA International, IBM Corporation and Capgemini.
- Minguez, J., F. Ruthardt, P. Riffelmacher, T. Scheibler, and B. Mitschang. 2010. Service-Based Integration in Event-Driven Manufacturing Environments. In *International Conference on Web Information Systems Engineering*, 295–308. https://doi.org/10.1007/978-3-642-24396-7_23.
- Monostori, L., B. Kádár, T. Bauernhansl, S. Kondoh, S. Kumara, G. Reinhart, O. Sauer, G. Schuh, W. Shin, and K. Ueda. 2016. Cyber-Physical Systems in Manufacturing. *CIRP Annals—Manufacturing Technology* 65 (2): 621–641. <https://doi.org/10.1016/j.cirp.2016.06.005>.
- O’Kane, J.M. 2014. *A Gentle Introduction to ROS*. Coleraine: Jason M O’Kane.
- Olson, D.L., B. Johansson, and R.A. De Carvalho. 2018. Open Source ERP Business Model Framework. *Robotics and Computer-Integrated Manufacturing* 50: 30–36. <https://doi.org/10.1016/j.rcim.2015.09.007>.
- Rehman, S., and V. Gruhn. 2018. An Effective Security Requirement Engineering Framework for Cyber-Physical Systems. *Technologies* 6 (3): 65. <https://doi.org/10.3390/technologies6030065>.
- Sauter, T. 2014. Fieldbus Systems Fundamentals. In *Industrial Communication Technology Handbook*, ed. R. Zurawski, 1–48. Boca Raton: CRC Press.
- Schleipen, M., A. Lüder, O. Sauer, H. Flatt, and J. Jasperneite. 2015. Requirements and Concept for Plug-and-Work. *at-Automatisierungstechnik* 63 (10): 801–820. <https://doi.org/10.1515/auto-2015-0015>.
- Steinberg, D.H., and S. Cheshire. 2005. *Zero Configuration Networking: The Definitive Guide*. O’Reilly Media. <https://doi.org/10.1515/auto-2015-0015>.
- Sztipanovits, J., X. Koutsoukos, G. Karsai, N. Kottenstette, P. Antsaklis, V. Gupta, J. Baras, and S. Wang. 2012. Toward a Science of Cyber-Physical System Integration. *Proceedings of the IEEE* 100 (1): 29–44. <https://doi.org/10.1109/JPROC.2011.2161529>.

- Vernadat, F. 2007. Interoperable Enterprise Systems: Principles, Concepts, and Methods. *Annual Reviews in Control* 31 (1): 137–145. <https://doi.org/10.1016/j.arcontrol.2007.03.004>.
- Vogel-Heuser, B., J. Lee, and P. Leitão. 2015. Agents Enabling Cyber-Physical Production Systems. *at-Automatisierungstechnik* 63 (10): 777–789. <https://doi.org/10.1515/auto-2014-1153>.
- Wagner, H., O. Pankratz, W. Mellis, and D. Basten. 2015. Effort of EAI Projects: A Repertory Grid Investigation of Influencing Factors. *Project Management Journal* 46 (5): 62–80. <https://doi.org/10.1002/pmj.21523>.
- Weber, S. 2004. *The Success of Open Source*. Cambridge: Harvard University Press.
- Weiss, G. 1999. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge: MIT Press.
- Wooldridge, M., and N.R. Jennings. 1995. Intelligent Agents: Theory and Practice. *The Knowledge Engineering Review* 10 (2): 115–152. <https://doi.org/10.1017/S0269888900008122>.
- Ye, X., and S.H. Hong. 2019. Toward Industry 4.0 Components: Insights into and Implementation of Asset Administration Shells. *IEEE Industrial Electronics Magazine* 13 (1): 13–25. <https://doi.org/10.1109/MIE.2019.2893397>.
- Zhang, Y., Z. Guo, J. Lv, and Y. Liu. 2018. A Framework for Smart Production-Logistics Systems Based on CPS and Industrial IoT. *IEEE Transactions on Industrial Informatics* 14 (9): 4019–4032. <https://doi.org/10.1109/TII.2018.2845683>.
- Zimmermann, H. 1980. OSI Reference Model-the ISO Model of Architecture for Open Systems Interconnection. *IEEE Transactions on Communications* 28 (4): 425–432. <https://doi.org/10.1109/TCOM.1980.1094702>.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

